Comput Syst Sci & Eng (2019) 6: 325–338

International Journal of
Computer Systems
Science & Engineering

Using Spatial Relations for Qualitative Specification of Gestures

Giuseppe Della Penna[1]*and **Sergio Orefice**[2]†

[1] *Department of Engineering, Computer Science and Mathematics, University of L'Aquila, Italy*
[2] *Department of Engineering, Computer Science and Mathematics, University of L'Aquila, Italy*

In this paper we present a qualitative spatial relation formalism capable to represent spatio-temporal knowledge. To this aim, we have added the notion of time within the formalisation of spatial relations in order to describe moving graphical objects. As a case study, we exploited this augmented formalism to support a qualitative specification of gestures, which are an increasingly relevant issue in the human-computer interaction field. The resulting technique provides gesture specification with a systematic and formal foundation.

Keywords: Human-Computer Interaction, Spatial Relations, Spatio-Temporal Knowledge, Gesture Specification

1. INTRODUCTION

A formal and systematic way to model spatial knowledge is a very important step to design formalisms and systems in contexts where visual representations are a key element, like in modern human-computer interaction. Commonly, for a formal discussion visual representations are conceived as a collection of graphical objects and a set of spatial relations arranging them in a two dimensional space. In this context, much research has been done in particular towards the formalisation of the spatial relations since spatial composition rules are fundamental in representing spatial knowledge and in designing visual systems. Indeed, a formal description of their structural characteristics is crucial to provide a systematic base and avoid ad-hoc implementations.

The literature presents a wide variety of spatial relation formalisms (see Section 2 for a survey of the classic approaches), most of which use a qualitative approach to represent spatial information. In this field, we have proposed a framework that includes common qualitative spatial relations such as topological (e.g., overlapping, adjacency, containment) and direction (e.g., left, up) relations, with the addition of new relations to model interconnections. This reflects the two basic modalities that can be used to compose graphical objects: by spatially arranging or by connecting them. In [1] we describe recent advances of this formalism whose main feature is that it is not domain-specific, but general and flexible enough to be used in a variety of contexts where the visual aspects, especially the spatial ones, are relevant. As an example, so far it has been profitably applied to the Information Extraction [2, 3, 4]. In those works we have developed a technique called VIE (*Visual Information Extraction*) which enables users to perform information extraction from visual documents driven by the visual appearance and the spatial arrangement of the information. Over the years, this technique has been used with success in several domains such as html documents, biomedical imaging, geospatial data and PDF files.

To open our formalism to other significant application fields, in this paper we extend it with the notion of time in order to apply spatial relations to moving graphical objects. As a case study, we show how to use these relations to support gesture-based interaction, which is a novelty in the classic spatial relation literature.

*giuseppe.dellapenna@univaq.it
†sergio.orefice@univaq.it

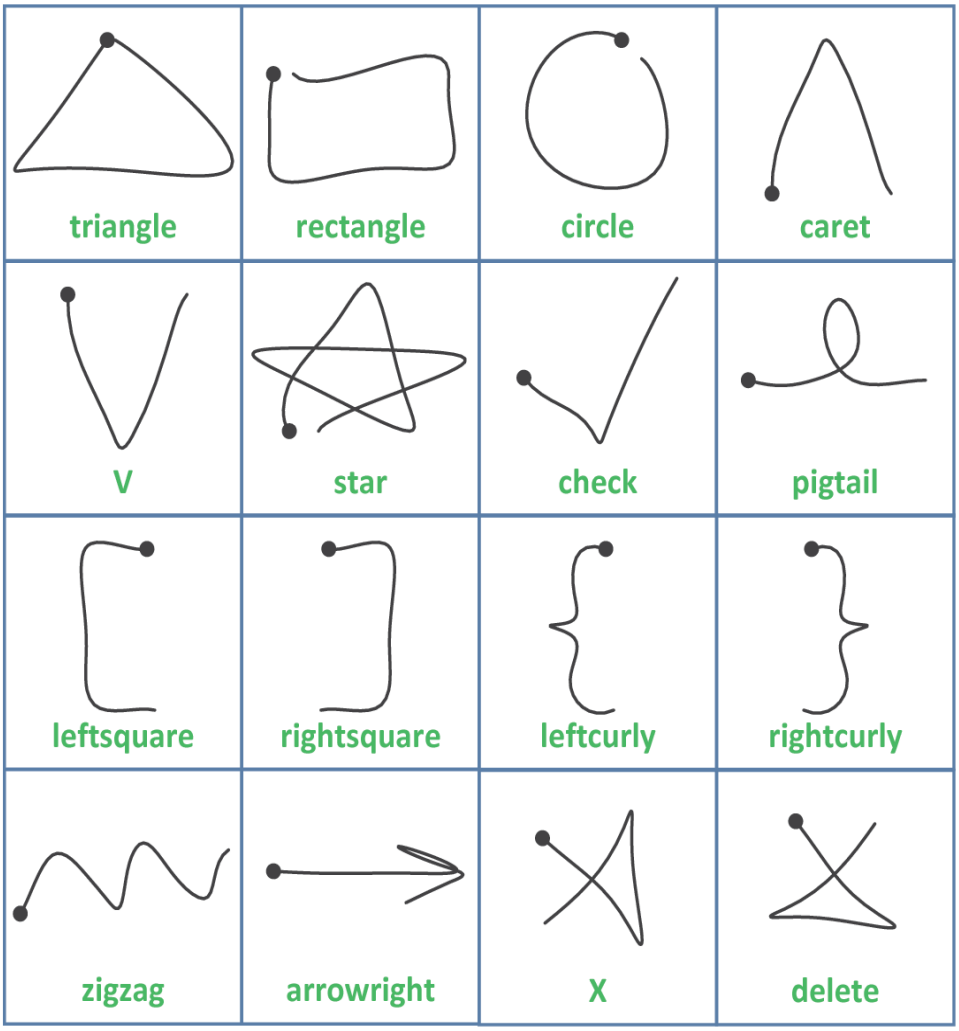


Figure 1 The "16 gesture types" of $1.

Gestures are 2D trajectories drawn by users with their fingers on a touch screen or with a pen, and are increasingly relevant to support designing user interfaces for mobile, tablet, large display, and tabletop computers. Along with the naturalness of gestures comes inherent ambiguity, making gesture specification and recognition a topic of great interest. However, in the current practice (see, e.g., [5, 6, 7, 8, 9, 10] for interesting applications in some specific fields), gesture encoding and recognition is commonly achieved through ad-hoc methods, sometimes even platform- or device-specific, but always relying on quantitative, geometric approaches, e.g., using numerical algorithms to match shapes or sets of points, which add further complexity to this task.

In this paper we focus on the gesture specification task and illustrate how to specify gestures by using a subset of our time-extended spatial relations (taken from [1]). Exploiting spatial relations makes our technique a novelty in the gesture specification field and provides two main advantages, as opposed to the above-cited ad-hoc approaches:

- the underlying theoretical spatial relation framework gives to the gesture specification a systematic and formal foundation;
- the gesture encoding style is very similar to the regular expression notation, making it easy to understand and manipulate.

To test our approach, in the paper we refer to the "16 gesture types" of $1 presented in [11] (see Figure 1), containing common gestures useful for making selections, executing commands, or entering symbols.

The paper is organised as follows. Related work is summarized in Section 2. In Section 3 we provide the theoretical features underlying our approach, recalling the notion of graphical object, describing the fragment of spatial relations we use for gesture specification and introducing the notion of time in order to apply spatial relations to moving objects. Then, in Section 4 we show how to use this augmented formalism for gesture specification and, as an example, we illustrate the encoding of all the "16 gesture types"of $1 in our formalism. Conclusions are outlined in Section 5, with a focus on the main further research.

2. RELATED WORK

Commonly, qualitative spatial relations are classified in two main classes, i.e., *topological* and *direction*, both describing where an

object is placed relative to another one. Topological relations describe qualitative spatial relations that are invariant under topological transformations such as translation, rotation and scaling. Examples of topological relations include adjacency, overlapping and containment. On the other hand, a direction relation still describes where an object is placed relative to another one, but without considering situations where the two objects overlap.

In the last decades, much research has been done in this field. In the following we briefly describe some of the main spatial relation formalisms presented in the literature.

Topological Spatial Relation Formalisms. N-intersection and RCC are the most widely recognised approaches to topological relations since most of the other approaches in the literature are based on these two formalisms.

In the *N-intersection* theory, an object is seen as a point set embedded in a specified space. In this framework, a region x is associated with three related point sets: the region *interior* x°, its *boundary* ∂x and its *exterior* x^-. Then, a relationship between any two regions x and y can be characterised by a matrix defining the intersections between each pair of the sets x°, ∂x, x^- and y°, ∂y, y^-. By considering only the first two sets (i.e., interior and boundary) we obtain the 4-intersection matrix [12]. Considering also the exterior leads to the well-known *9-intersection model*, which can be used to define the relationships between all combinations of lines, points and regions [13]. Even if many other formalisms derived from the N-intersection theory have been presented (see, e.g., [14] or [15]), the 9-intersection model is the most used to reason on the topological relationships between spatial objects.

The *Region Connection Calculus* (RCC, see, e.g., [16]), unlike the 9-intersection approach, takes regions rather than points as a fundamental notion. Relations are defined in the RCC formalism starting from the base *connected* relationship $C(x, y)$, which holds if and only if the regions x and y share a common point. A very large number of relations can be derived from $C(x, y)$. In particular, the eight relations *disconnected* $DC(x, y)$, *equal* $EQ(x, y)$, *partially overlapping* $PO(x, y)$,, *externally connected* $EC(x, y)$,, *tangential proper part* $TPP(x, y)$ (with its inverse $TPPi(x, y)$) and *non-tangential proper part* $NTPP(x, y)$ (with its inverse $NTPPi(x, y)$) are closed under weak composition and form a Jointly Exhaustive and Pairwise Disjoint (JEPD) set of relations known as the *RCC8* formalism, (which has been more exhaustively investigated in, e.g., [17]), where two regions stand to each other in exactly one of these relations. However, by adding new primitive relations and/or further combining the derived ones, more complex sets of spatial relations can be derived from RCC, as in [18].

Direction Spatial Relation Formalisms. Depending on the dimension of the objects involved, direction calculi can be divided into *point-based*, where objects are simplified into points, or *extended-object* based, where object have a specific *shape*.

Two well-known examples of point-based direction calculi are the *Oriented Point Relation Algebra* (OPRA, see, e.g., [19, 20]), where the notion of point is extended to *oriented point* (defined as a (point, direction) pair), i.e., an abstraction of object with an intrinsic direction, and the *Ternary Point Configuration Calculus* (TPCC, see, e.g., [21]), which uses *ternary relations*, in contrast with the more common binary ones. In particular, OPRA can be seen as an extension of another binary point-based calculi, *the Cone-Shaped Calculus* [22], whose relations are based on the eight disjoint sectors (*north, south, east, west, northwest, northeast, southwest, southeast*) of the space divided by lines going through the reference point. In addition, OPRA has an adjustable granularity, i.e, it allows an arbitrary number of oriented lines through the reference and target points, which define finer relations. On the other hand, TPCC derives its direction information from the double-cross calculus [23] and introduces the notion of *distance*. The resulting calculus has 27 relations obtained as combinations of the space partitions *front*, *back*, *left*, *right*, *straight*, *distant* and *close*. The TPCC relations are invariant when all points are mapped by rotations, scalings or translations.

Extended-object based calculi are more complex, since extended objects have a *shape*. Thus, to simplify the process, *minimal bounding rectangles* are often used as an approximation of the actual objects.

The *Cardinal Direction Calculus* [24] is the most well-known binary direction relation calculus. An arbitrary basic CDC relation is a binary relation involving a target object and a reference object, and a non-empty subset of the nine atomic relations N, NW, NE, W, O, E, S, SW, SE, corresponding to the possible intersections of sub regions of the target object with the 3×3 *direction matrix* [25] of the reference object. The CDC has been also the subject of many extensions and theoretical works. As an example, [26] merges the rectangle algebra into the CDC, producing a tractable subset of 36 rectangular relations. However, it is worth noting that the CDC cannot handle direction information between overlapping and contained regions properly.

Another well-known binary direction relation calculus is the *Rectangle Algebra* (RA) [27], an extension of the *Interval Algebra* [28]. Objects in this formalism are restricted to be *rational rectangles*, i.e, rectangles whose sides are parallel to the axes of some orthogonal basis in a 2-dimensional Euclidean space. Relations between these objects are the 13×13 pairs of atomic relations which can hold between two rational intervals, and can be used to express directional relations but also topological relations such as disjoint and overlap. Spatial information is represented by *spatial constraint networks*, i.e., constraint satisfaction problems where variables represent rational rectangles and constraints are relations. This model, though restrictive, is sufficient for applications in domains like architecture or GIS.

Finally, the *Objects Interaction Matrix* (OIM, see, e.g., [29]) aims to overcome a limitation of other direction calculi, including advanced ones as the CDC, which may give erroneous or counterintuitive results when applied to objects with a complex shape like, e.g., the regions of a geographic map. OIM consists of a two-phase model. The former is a *tiling phase*, where a tiling strategy determines the zones belonging to the nine cardinal directions around each individual object and intersects them, creating a bounded grid called *interaction grid*. The objects interaction matrix is then used to store the information about which object intersects each grid cell. Then, in the *interpretation phase*, a well-known interpretation method

(e.g., the nine directions defined by the 9-interaction model) is used on the matrix and determines the detail of cardinal directions between the objects. However, as proved in [30], when representing the cardinal direction of two regions by a consistent pair of direction-relation matrices (DRMs), then the OIM is actually not so different from the DRM, and thus essentially the same as CDC.

Further references to topological and direction spatial relation formalisms can be found in the extended survey presented in [31].

The spatial relation literature also contains some works about qualitative spatio-temporal relations, which introduce the notion of time in order to model the temporal evolution over time of spatial aspects. Among the others, e.g., [32] analyses the domains of interval temporal relations proposing a set of algorithms to derive relations between intervals. In particular, such algorithms are extended for objects in an arbitrary n-dimensional space, so that presentation layouts in 2D space e can be addressed, [33] introduces a two-dimensional logic capable of describing topological relationships that change over time and containing also various temporal extensions of the spatial logic RCC8, whereas [34] presents a spatio-temporal querying and retrieval system to interactively build subsequent spatio-temporal queries using features of gaming controllers. On the other hand, trends on qualitative spatio-temporal reasoning can be found in, e.g., [35, 36, 37].

However, to the best of our knowledge, none of the spatio-temporal relation approaches has been applied to the specification of gestures which in the current practice is commonly achieved using quantitative, geometric approaches. Among these, in the following we briefly focus on the formalisms that mainly inspired our approach to the gesture specification.

$1 [11] is a simple gesture recognizer based on gesture templates which are suitably compared to the user gesture using a path distance algorithm. In particular, the gesture is first resampled, rotated, scaled and moved to match it as much as possible with each template, and then the path distance is obtained as the average distance between the gesture points and the corresponding template points. The template associated with the lower path-distance is chosen as the recognized gesture.

Protractor [38] is an evolution of $1 with simplified scaling and rotation techniques. Indeed, Protractor adds a pre-processing step which resamples the gesture into a fixed number of equidistant points and translates them in order to position the gesture centroid on (0,0). In this way, it removes noise such as different drawing speed, different on-screen positions and gesture orientation, etc. and transforms the gesture into a uniform vector representation.

$P [39] is a further development of the "$ family". Indeed, other $-recognizers consider the order, direction, and number of points in the sampled gesture, which however require much more memory and time to be processed and make the gesture recognition dependant on the input direction. To overcome these limitations, $P considers the gesture as a point cloud (i.e., without order information).

Finally, *PolyRec* [40] is a scale- and rotation-invariant gesture recognizer exploiting a peculiar sampling policy. Indeed, it does not sample the gesture points with a fixed distance, but extracts only a small number of "dominant points", i.e., the points where the gesture shows a big variation in its curvature, and uses these points as the vertices of a polyline which encodes the main movements of the gesture. When comparing two gestures, the two corresponding polylines are adjusted, possibly adding new vertices, in order to have the same number of segments, and the corresponding segments are compared in pairs in order to obtain the overall distance between the two gestures.

3. THEORETICAL BACKGROUND

In this Section we present the main theoretical concepts used in our gesture specification approach.

3.1 Graphical Objects

In the following we recall the notion of graphical object as defined in [1].

A graphical object is formally defined as pair $O = (C, A)$, where C denotes the set of all the points $p \in \mathbb{R}^2$ forming the external contour of O (which is disjoint from its internal area), and A is the set of the object *attributes*. Each attribute is a pair (a, v), where a is a property name and v its value.

We consider only graphical objects which do not contain *holes* and have a contour that can be modelled as a *closed curve* without self loops (*simple curve*).

Conventionally, we shall write $p \in C$ to indicate a generic point of C and p_x, p_y, to indicate the x and y-coordinate, respectively, of p.

3.2 Disjoint Spatial Relations

In this Section we first recall the class of *disjoint spatial relations* taken from the overall formalism presented [1]. This class contains four distinct spatial relations, namely, *UP*, *DOWN*, *LEFT* and *RIGHT*, that are sufficient to model any disjoint spatial arrangement between two graphical objects.

In order to define them, we need to introduce the notation $y_{UM(C)}$ to represent the highest y coordinate of the points in the contour C, $y_{DM(C)}$ to represent the lowest y coordinate of the points in the contour C, $x_{LM(C)}$ to represent the lowest x coordinate of the points in the contour C and $x_{RM}(C)$ to represent the highest x coordinate of the points in the contour C. In particular, $y_{UM(C)}$ is the y coordinate of each point in $UM(C) = \{p \in C | \forall p' \in C, p'_y \leq p_y\}$ (in other words, $UM(C)$ denotes the set of upmost points of the contour C). The three other points can be similarly defined. Note that, in our formalism, we refer to the canonical orientation of the Cartesian axes (i.e., the x coordinate increases rightwards, and the y one increases upwards).

Then, given two graphical objects $O = (C, A)$ and $O' = (C', A')$, the four disjoint spatial relations are defined as in Figure 2. Intuitively, the *UP* relation models a spatial arrangement between O and O' whenever the graphical object O' is completely (both internal points and external contour) below the graphical object O.

It is also possible both to define *weak* versions of each spatial relation and to *compose* spatial relations among them. For example, the *WEAK-UP* disjoint spatial relation can be formalised as follows by appropriately relaxing the involved constraints:

$O\ UP\ O'$	$\Leftrightarrow$	$y_{UM(C')} \leq y_{DM(C)}$
$O\ DOWN\ O'$	$\Leftrightarrow$	$y_{UM(C)} \leq y_{DM(C')}$
$O\ LEFT\ O'$	$\Leftrightarrow$	$x_{RM(C)} \leq x_{LM(C')}$
$O\ RIGHT\ O'$	$\Leftrightarrow$	$x_{RM(C')} \leq x_{LM(C)}$

Figure 2 Definition of disjoint spatial relations.

$O\ \ STRAIGHT - UP(x_1, x_2)\ \ O'$ $\Leftrightarrow y_{DM(O)} >= y_{UM(O')} \wedge x_{LM(O)} >= x_{LM(O')} \quad x_1 \wedge$ $x_{RM(O)} <= x_{RM(O')} + x_2$
$O\ \ STRAIGHT - LEFT(y_1, y_2)\ \ O'$ $\Leftrightarrow x_{RM(O)} <= x_{LM(O')} \wedge y_{UM(O)} <= y_{UM(O')} + y_1 \wedge$ $y_{DM(O)} >= y_{DM(O')} + y_2$
$O\ \ STRAIGHT - DOWN(x_1, x_2)\ \ O'$ $\Leftrightarrow y_{UM(O)} <= y_{DM(O')} \wedge x_{LM(O)} >= x_{LM(O')} + x_1 \wedge$ $x_{RM(O)} <= x_{RM(O')} + x_2$
$O\ \ STRAIGHT - RIGHT(y_1, y_2)\ \ O'$ $\Leftrightarrow x_{LM(O)} >= x_{RM(O')} \wedge y_{UM(O)} <= y_{UM(O')} + y_1 \wedge$ $y_{DM(O)} >= y_{DM(O')} + y_2$

Figure 3 Definition of straight disjoint spatial relations.

$$O\ WEAK - UP\ O' \Leftrightarrow y_{UM(C')} \leq y_{UM(C)}$$

In other words, in this specific case it is not required for O' to be *completely* below O (as in the case of *UP*) but also *partially*, i.e., the uppermost point of O' must be below the uppermost point of O. The other three weak relations (i.e, *WEAK-DOWN*, *WEAK-LEFT* and *WEAK-RIGHT*) can be similarly defined.

On the other hand, we may use common logical connectives to compose spatial relations in order to obtain derived relations like, e.g., $UP \wedge RIGHT$, which models the spatial arrangement where an object is both above and on the right with respect to another one.

To effectively apply our formalism to the specification of gestures, we also add four new spatial relations to the disjoint class, namely the *straight* disjoint spatial relations *STRAIGHT–UP, STRAIGHT–LEFT, STRAIGHT–DOWN*, and *STRAIGHT–RIGHT*.

These relations describe spatial arrangements where two objects are "aligned", within a given threshold, in one of the four cardinal directions, and can be defined as shown in Figure 3.

Moreover, to improve the notation that will be used for gesture specification, we introduce the notion of *inverse relation*, useful for description purposes and denoted as $\overleftarrow{R}$. Actually, for each spatial relation R, $O\ \overleftarrow{R}\ O'$ holds if and only if $O'RO$ holds.

3.3 Time

In this Section we introduce the basic concepts allowing to extend our spatial relation formalism with the notion of time.

Timed graphical objects. In a spatio-temporal context, graphical objects become dynamic elements, whose configuration changes over time. The particular configuration of a graphical object can be formalised as follows.

Definition 1 [Object instantaneous configuration] Let O be a graphical object and $T = \{t_0, t_1, \ldots, t_n\}$ be a discretised timeline. We write O^i, with $i \in [0 \ldots n]$ to denote the instantaneous configuration of the object O at the instant t_i of the timeline.

In general, the object instantaneous configuration may include both its contour and its attributes, i.e., $O^i = (C^i, A^i)$. However, as we are interested in specifying gestures, we will restrict our attention to *object movements* only. To this aim, instantaneous configurations of the same object will have the same attributes (i.e., attributes are immutable over time) and contours cannot change shape, i.e., when they move all the included points are translated by the same amount. Formally, if $O^i = (C^i, A)$ then we have $O^{i+1} = (C^{i+1}, A)$ where $C^{i+1} = C^i + \vec{v}$, i.e., every point in C^i is translated by vector $\vec{v}$.

Spatial relations applied to timed objects. The spatial relations defined in Section 3.2 can be naturally applied to the object instantaneous configurations, e.g., if O and O' are graphical objects, R is a spatial relation and $t_i, t_j \in T$ we can write $O^i\ R\ O'^j$ if and only if the position of object O at the instant t_i is in relation R with the position of the object O' at the instant t_j.

In order to inject the notion of time directly in the spatial relation notation, let us define the *AT* operator as follows.

Definition 2 [AT spatial relation operator] Let R be a spatial relation and $t_i, t_j \in T$. Then, $AT(R, i, j)$ is a spatial relation such that, for any graphical objects O and O',

$$O\ AT(R, i, j)\ O' \Leftrightarrow O^i R\ O'^j$$

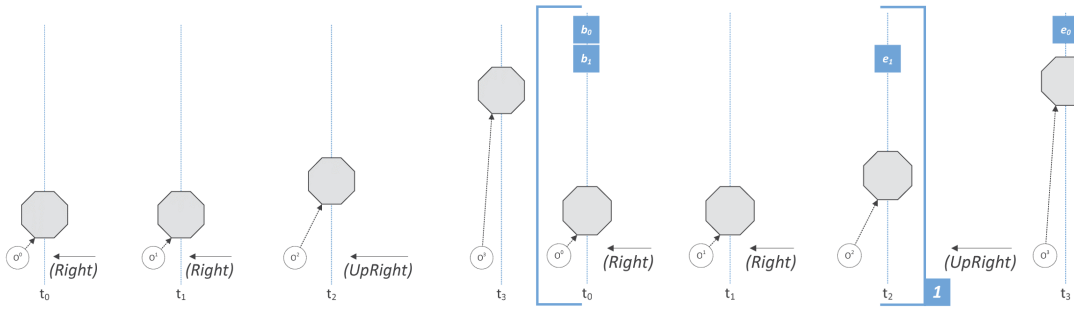


Figure 4 Graphical object trajectory.

Graphical object trajectory. The central concept of our spatio-temporal formalism is the notion of *trajectory*, i.e., the movement on the plane of a graphical object.

Actually, it is possible to capture the notion of *object movement* taking into account the relations holding among consecutive configurations of the same graphical object. As an example, we may write $O\ AT(LEFT, 0, 1)\ O$) to indicate that, at the initial time t_0, the object O is on the left of its position at time t_1, in other words it *moved to the right*. However, using the *LEFT* relation to indicate a right-wise movement may be counterintuitive, whereas the use of the *RIGHT* relation would make the previous statement more clear. Then, we may exploit the inverse *RIGHT* relation $\overleftarrow{RIGHT}$ writing simply $O\ AT(\overleftarrow{RIGHT}, 0, 1)\ O$ instead of $O\ AT(LEFT, 0, 1)\ O$.

Formally, the trajectory of a graphical object can be defined as follows.

Definition 3 [Object trajectory] Let $T = \{t_0, t_1, \ldots, t_n\}$ be a discretised timeline, O a graphical object and $\mathcal{R}$ a set of spatial relations. A trajectory, denoted by Γ, of O from instant $t_h \in T$ to instant $t_k \in T$ with $k > h$ can be written as

$$\begin{aligned}\Gamma = & O\ AT\ (REL_1, h, h+1)\ O \wedge \\ & O\ AT\ (REL_2, h+1, h+2)\ O \wedge \\ & \ldots \wedge O\ AT\ (REL_{k-h}, k-1, k)\ O\end{aligned}$$

where each REL_i is a spatial relation from $\mathcal{R}$.

Of course, in the practice, REL_i will always be an inverse relation to simplify the notation as discussed above. Moreover, when the graphical object involved in the trajectory is clear from the context, to encode trajectories in a more compact way we shall rewrite the expression in the following simpler form

$$\begin{aligned}\Gamma = & AT\ (REL_1, h, h+1)\,, \\ & AT\ (REL_2, h+1, h+2)\,, \\ & \ldots, AT\ (REL_{k-h}, k-1, k)\end{aligned}$$

where the graphical object has been omitted and the symbol $\wedge$ has been replaced by a comma.

Finally, since in trajectories the time always flows linearly, one instant per relation, if the start time h is clear from the context, the expression can be further simplified as

$$\Gamma = REL_1, REL_2, \ldots, REL_{k-h}$$

by completely omitting the time part.

As an example, the expression $\Gamma = \overleftarrow{RIGHT}, \overleftarrow{RIGHT}, \overleftarrow{UPRIGHT}$ (where, intuitively, *UPRIGHT* represents the derived relation $UP \wedge RIGHT$) models the trajectory depicted in Figure 4(a), where the graphical object O first moves to the right for two instants and then right-upwards.

Sub-trajectories. Specific fragments of a trajectory can be identified by enclosing them in numbered square brackets, e.g., we can write $\Gamma = [\overleftarrow{RIGHT}, \overleftarrow{RIGHT}]_1, \overleftarrow{UPRIGHT}$ to capture the first part of the overall trajectory as the sub-trajectory Γ_1. As a convention, the index 0 is used to address the whole trajectory, i.e., $\Gamma_0 = \Gamma$.

Moreover, the first and last time instants occurring in a sub-trajectory Γ_i are denoted by b_i and e_i, respectively. As an example, in the trajectory Γ depicted in Figure 4(b) we have that $b_1 = 0$ and $e_1 = 2$. Note that, by abuse of notation, we shall also write b_i for O^{b_i} and e_i for O^{e_i}, e.g., in the above example, O_1 can be used to indicate O^2, too.

Finally, when spatial relations are repeated more times, we use the "plus" operator $+$ to denote a non empty sequence of relations. For example, the trajectory Γ above may be rewritten as $\Gamma = [\overleftarrow{RIGHT}+]_1, \overleftarrow{UPRIGHT}$ in our formalism. This notation is feasible when specifying gestures, because in that context the goal is to encode a continuative movement in a specific direction (i.e., an unspecified length sequence of the same spatial relation) as a whole, rather than as a discrete sequence of steps.

In this way, we end with a trajectory encoding style which is very similar to well-known formal notations as regular expressions, making it more easy to understand and manipulate.

4. SPECIFYING GESTURES

In order to specify gestures, we represent the finger moving on the touch screen as a *timed graphical object*. Of course, we assume that such object has a size consistent with the area activated by the finger pressure on the screen. Moreover, since gestures are continuous movements, we need a suitable time discretisation strategy to sample the finger position in order to encode its movement as a trajectory over a discretised timeline. In particular, we sample the finger position every n pixels, obtaining a spatially meaningful sequence of instantaneous configurations, through the use of variable-length temporal intervals (i.e., not fixed-length, which would make the encoding too much influenced by the execution time of the gesture). In this way, any gesture will be encoded as a specific trajectory of a timed graphical object (see Figure 5(a) for an intuitive sketch). In our encoding formalism we will also exploit the notion

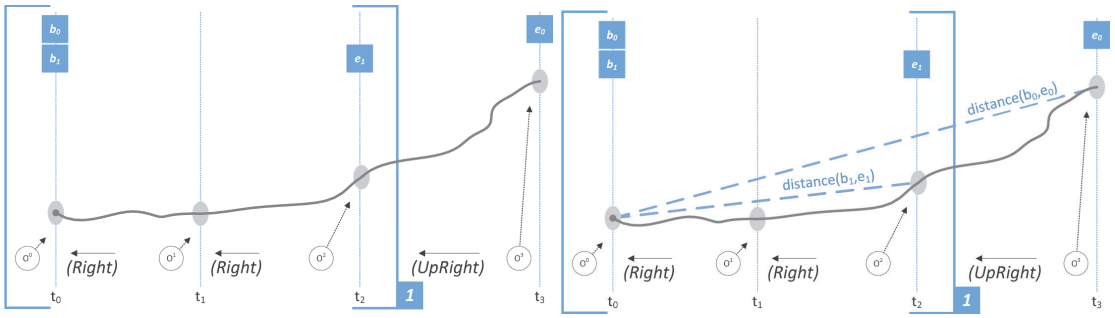
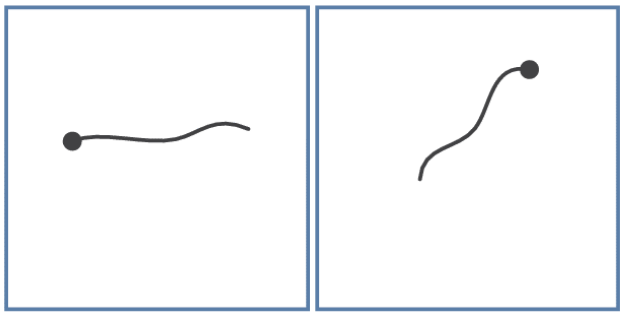
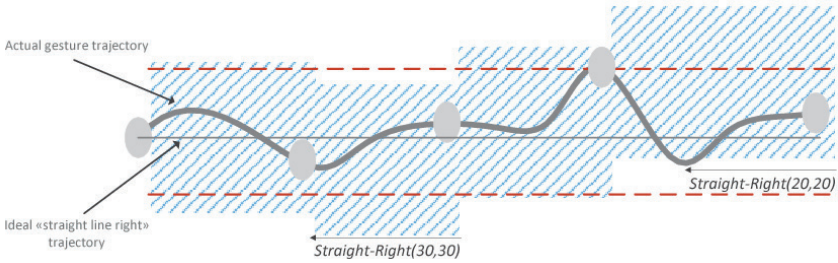


Figure 5 A gesture sampled as a timed graphical object trajectory.



(a) LineRight (b) LineDownLeft



(c) Detail of a possible LineRight valid trajectory

Figure 6 Examples of basic gestures.

of sub-trajectory first and last time instant (as defined in the previous Section) by comparing the instantaneous configurations corresponding to those instants through the use of spatial relations together with an appropriate function that measures the distance between two graphical objects O, O' and it is formally defined as follows:

$$distance\left(O, O'\right) = min_{p \in C, q \in C'} \| p - q \|$$

As an example, with reference to the trajectory shown in Figure 5(b), we may write expressions like $b_1, \overleftarrow{RIGHT} e_1 \wedge distance(b_1, e_1) \leq 0.2$.

In general, such kind of expressions allow us to refine the basic spatial gesture specification provided by the graphical object trajectory as explained above, e.g., verifying whether the beginning and ending points of the trajectory are close, or comparing the lengths of two sub-trajectories.

4.1 Basic Gestures

In the following we present some examples showing how to encode basic gestures like "LineRight" or "LineDownLeft". These gestures are not included in the "16 gesture types" of \$1, but are useful to support a simple composition of them. Indeed, *gesture compositionality* is a key feature characterizing our approach and allowing gesture specification reuse, as it will be clear from the examples of Section 4.2.

Cardinal gestures. To begin, let us define in detail one of these gestures, for example the *LineRight* gesture which is shown in Figure 6(a), where the dot represents the starting point of the gesture trajectory.

$$LineRight = \overleftarrow{STRAIGHT - RIGHT}(30, 30) + \\ with\ b_0\ \overleftarrow{STRAIGHT - RIGHT}(20, 20)\ e_0$$

In this case, the trajectory models an "almost straight" right-wise movement, i.e., allowing a suitable amount of "noise" (30 pixel in the specification), with respect to the ideal straight line, between each trajectory object instantaneous configuration and its next one, as sketched in Figure 6(c). Moreover, the trajectory constraint requires the beginning and the end of the line (i.e., the first and the last object instantaneous configuration in the sub-trajectory 0) to be better horizontally aligned, e.g., with a smaller tolerance of 20 pixel.

This kind of notation also introduces in our gesture specification formalism a certain degree of *noise tolerance*, in

the sense that the formulas may tolerate unaligned points, within the ranges defined by both the relations and constraints.

The other basic cardinal gestures *LineLeft*, *LineUp* and *LineDown* can be defined similarly as follows:

$$\begin{aligned} LineLeft &= \overleftarrow{STRAIGHT-LEFT}(30,30)+ \\ &\quad with\ b_0\ \overleftarrow{STRAIGHT-LEFT}(20,20)\ \ e_0 \\ LineUp &= \overleftarrow{STRAIGHT-UP}(30,30)+ \\ &\quad with\ \ b_0\ \ \overleftarrow{STRAIGHT-UP}(20,20)\ \ e_0 \\ LineDown &= \overleftarrow{STRAIGHT-DOWN}(30,30)+ \\ &\quad with\ b_0\ \overleftarrow{STRAIGHT-DOWN}(20,20)\ e_0 \end{aligned}$$

It is worth noting that these gestures are also useful stand alone since they correspond to the common "swipe" gestures used in the graphical interfaces.

Diagonal gestures. Again, let us define in detail one of these gestures, for example the *LineDownLeft* gesture, which is shown in Figure 6(b).

$$\begin{aligned} LineDownLeft &= \overleftarrow{WEAK-DOWN-LEFT}+ \\ &\quad with\ \ b_0\ \ \overleftarrow{DOWN-LEFT}\ \ e_0 \end{aligned}$$

where the spatial relation $WEAK-DOWN-LEFT$ is composed by $WEAK-DOWN \wedge WEAK-LEFT$, whileas $DOWN-LEFT$ is composed by $DOWN \wedge LEFT$.

This time, the trajectory models an "almost straight" down-left movement with a greater noise tolerance (motivated by the larger approximation typical of non-cardinal movements) obtained exploiting weak relations which prevent movements in up or left direction.

Moreover, the trajectory constraint ensures a real diagonal movement by requiring the beginning and the end of the line to be in the (stronger) $\overleftarrow{DOWN-LEFT}$ relation.

The other diagonal gestures *LineUpRight*, *LineUpLeft* and *LineDownRight* can be defined similarly as follows:

$$\begin{aligned} LineUpRight &= \overleftarrow{WEAK-UP-RIGHT}+ \\ &\quad with\ b_0\ \overleftarrow{UP-RIGHT}\ e_0 \\ LineUpLeft &= \overleftarrow{WEAK-UP-LEFT}+ \\ &\quad with\ b_0\ \overleftarrow{UP-LEFT}\ e_0 \end{aligned}$$

$$\begin{aligned} LineDownRight &= \overleftarrow{WEAK-DOWN-RIGHT}+ \\ &\quad with\ b_0\ \overleftarrow{DOWN-RIGHT}\ e_0 \end{aligned}$$

4.2 $1 16 gestures types mapping

In this Section we show how to encode the "16 gesture types" of $1 (see Figure 1) within our formalism. In order to do this, we will exploit the compositionality of our framework through the basic gestures we have just defined.

triangle. The *triangle* gesture is defined as follows:

$$\begin{aligned} triangle = LineDownLeft, [LineRight]_1, LineUpLeft \\ with\ \ distance(b_0,e_0) \leq 0.2 \cdot distance(b_1,e_1) \end{aligned}$$

This definition contains the triangle trajectory, decomposed in three basic sub-gestures, together with an appropriate constraint. These sub-gestures are expressed in the overall trajectory notation as "macros" which are used to appropriately accomplish the gesture compositionality. These macros are suitably expanded in order to obtain an actual trajectory and a corresponding set of constraints. In the above *triangle* gesture, the resulting fully-expanded specification is the following:

$$\begin{aligned} &\left[\overleftarrow{WEAK-DOWN-LEFT}+\right]_1, \\ &\left[\overleftarrow{STRAIGHT-RIGHT}(30,30)+\right]_2, \\ &\left[\overleftarrow{WEAK-UP-LEFT}+\right]_3 \\ triangle = with\ b_1\ &\overleftarrow{DOWN-LEFT}\ e_1 \wedge \\ b_2\ &\overleftarrow{STRAIGHT-RIGHT}(20,20)\ e_2 \wedge \\ b_3\ &\overleftarrow{UP-LEFT} e_3 \wedge \\ &distance(b_0,e_0) \leq 0.2 \cdot distance(b_2,e_2) \end{aligned}$$

Here, the first three constraints derive from expanding the three basic sub-gestures, whereas the last one (i.e., $distance(bb_0,e_0) \leq 0.2 \cdot distance(b_2,e_2)$) requires the beginning and end positions of the overall gesture, which should ideally coincide to form the upper angle of the triangle shape, to be "close enough", that is with a distance (i.e., B in Figure 7(a)) which is no more than 20% of the triangle base (i.e., B in that figure). This makes the gesture tolerance scalable with respect to the actual size of the drawn triangle.

This example is suitable to show the qualitative nature of our specification formalism, which is not affected by specific numerical measures. Indeed, the formalism is itself inherently *scale invariant*, unlike the traditional quantitative approaches, where the scale invariance (when considered) is obtained through numerical algorithms trying to adapt the gesture sample to its actual instances.

It is worth noting that this kind of specification typical of our formalism also makes it *rotation invariant*, in the sense that the overall specification is still feasible even when the gesture is slightly rotated within the limits set by the tolerances included in the formulas.

rectangle. The *rectangle* gesture is defined as follows:

$$\begin{aligned} rectangle = LineDown, \left[LineRight\right]_1, LineUp, LineLeft \\ with\ distance(b_0,e_0) \leq 0.2 \cdot distance(b_1,e_1) \end{aligned}$$

As in the case of the *triangle* gesture, here the constraint (i.e., $distance(b_0,e_0) \leq 0.2 \cdot distance(b_1,e_1)$) ensures that the two edges of the gesture are close enough, i.e. their distance is again no more than 20% with respect to the rectangle size (estimated through the length of its base). In particular, to model the specific kind of rectangle in the $1 table (where the base is the longer side) it would be enough to add a simple constraint between the first two sub-gestures of the formula (i.e., $LineDown$ and $LineRight$) requiring that the length of the former is about half of the latter one length.

It is worth noting that, by adding a further simple constraint imposing that all the sides of the rectangle have about the same length, the same formula could be used to define a square-like gesture.

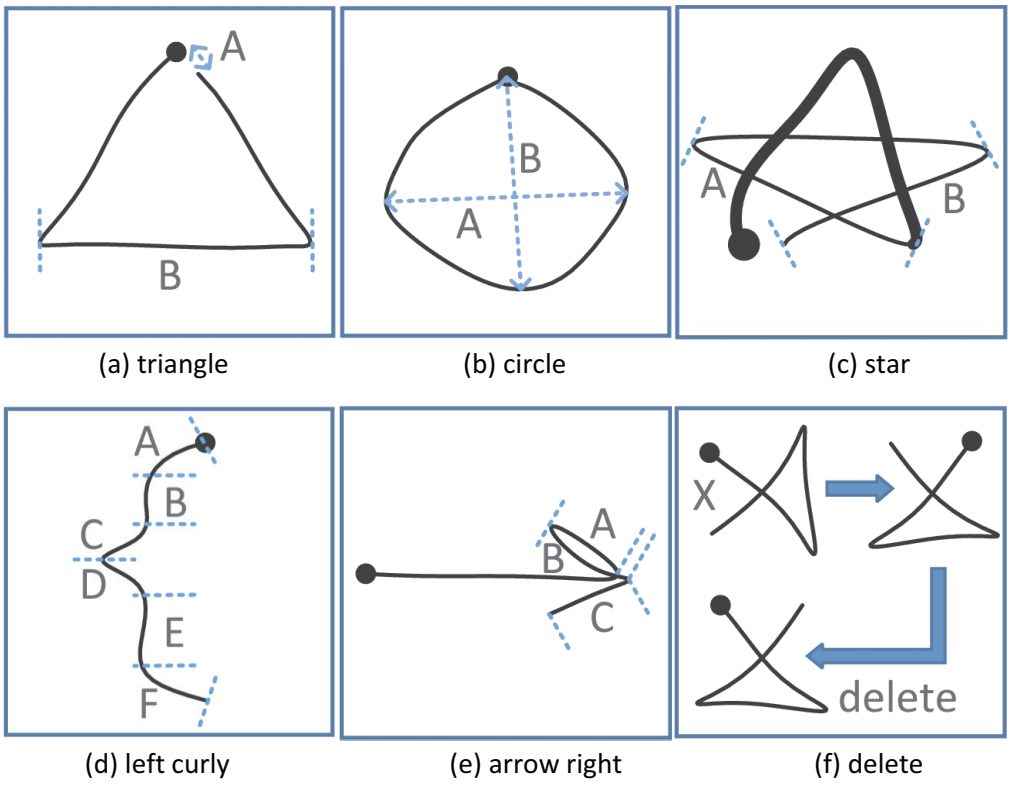


Figure 7 Specification details for gestures *triangle*, *circle*, *star*, *leftcurly*, *arrowright* and *delete*.

circle. The *circle* gesture is defined as follows:

$$\begin{aligned} &\left[LineDownLeft\right]_1, \left[LineDownnRight\right]_2 \\ &\left[LineUpRight\right]_3,\ LineUpLeft \\ circle = &\ with\ distance\,(b_0, e_0) \leq 0.2 \cdot distance\,(b_1, e_1) \wedge \\ &|distance\,(b_1, e_2) - distance\,(e_1, e_3)\,| \\ &\leq 0.2 \cdot distance\,(b_1, e_2) \end{aligned}$$

We assume that the circle is represented as diamond-like shape, as it actually happens when the user traces it on the screen.

In the formula, the former constraint (i.e., $distance(b_0, e_0) \leq 0.2 \cdot distance(b_1, e_1)$) requires the gesture trajectory to be approximatively closed, similarly to the previous cases, whereas the latter one (i.e., $|distance(b_1, e_2) - distance(e_1, e_3)| \leq 0.2 \cdot distance(b_1, e_2)$) requires its width and height to be nearly identical. To this aim, the last constraint simply checks that the difference between the gesture "width" and "height" (i.e., A and B in Figure 7(b)) does not exceed 20% of the gesture "size", here approximated by its width.

It is worth noting that, setting a different width/height ratio, the same formula could be used to define an ellipse-like gesture.

caret and V. The *caret* and V gestures are defined as follows:

$$\begin{aligned} &\left[LineUpRight\right]_1, \left[LineDownRight\right]_2 \\ caret = &\ with\ |distance\,(b_1, e_1) - distance\,(b_2, e_2)\,| \\ &\leq distance\,(b_1, e_1) \cdot 0.2 \\ &\left[LineDownRight\right]_1, \left[LineUpRight\right]_2 \\ V = &\ with\ |(distance(b_1, e_1) - distance(b_2, e_2)| \\ &\leq distance\,(b_1, e_1) \cdot 0.2 \end{aligned}$$

In these formulas the constraint (i.e., $|distance(b_1, e_1) - distance(b_2, e_2)| \leq distance(b_1, e_1) \cdot 0.2$) is the same and requires that the two lines composing both the gestures are of nearly identical length, in the sense that their difference is at most 20%.

These gestures suitably show how, in our formalism, deriving symmetric gestures (with respect to the horizontal and/or vertical axis) is very natural, thanks to the inherent *symmetry* present in the set of disjoint spatial relations of Section 3.2 (e.g., as in the case of UP and $DOWN$ that are symmetric with respect to the x axis). This symmetry was already evident in the basic gestures where, for example, by swapping the relations *WEAK–UP–RIGHT* and *UP–RIGHT* with *WEAK–DOWN–RIGHT* and *DOWN–RIGHT*, respectively, the basic diagonal gesture *LineUpRight* becomes its x-symmetric *LineDownRight*. Then, since the V gesture is essentially a caret mirrored on the x axis, the formula for the V gesture is simply obtained from the caret one by appropriately inverting the basic gestures composing it.

star. In the definition of the *star* gesture we can further take advantage of the compositionality of our formalism by exploiting the caret gesture specification as a component of this more complex gesture:

$$\begin{aligned} &caret, \left[LineUpLeft\right]_1, LineRight, \\ &\left[LineDownLeft\right]_2 \\ star = &\ with\ distance\,(b_0, e_0) \leq 0.2 \cdot distance\,(b_1, e_1) \wedge \\ &|distance\,(b_1, e_1) - \text{distance}\,(b_2, e_2)\,| \\ &\leq dis\,\text{tan}\,ce\,(b_1, e_1) \cdot 0.2 \end{aligned}$$

Here, it is worth noting that the initial fragment of the star (i.e., the one starting from the dot point as depicted in Figure 7(c)) is obtained reusing the caret specification.

In the formula, the former constraint (i.e., $distance(b_0, e_0) \leq 0.2 \cdot distance(b_1, e_1)$) requires again the gesture trajectory to be approximatively closed, whereas the latter one (i.e., $|distance(b_1, e_1) - distance(b_2, e_2)| \leq distance(b_1, e_1) \cdot 0.2$) requires the two gesture diagonal lines not belonging to the caret (i.e., lines A and B in Figure 7(c)) to have a nearly identical length. We are confident that these constraints are sufficient to specify an adequately regular shape, whereas additional constraints (e.g., length similarity between all the diagonal lines composing the gesture) would even make the gesture trajectory hard to trace for the user in the practice.

check. The *check* gesture is defined as follows:

$$check = \begin{array}{l} \left[LineDownRight\right]_1, \left[LineUpRight\right]_2 \\ with\ distance\,(b_1, e_1) \leq 0.6 \cdot distance\,(b_2, e_2) \end{array}$$

In this simple gesture, we only need to use a constraint (i.e., $distance(b_1, e_1) \leq 0.6 \cdot distance(b_2, e_2)$) to have the right proportion (0.6 in the formula) between the two sides of the check mark and make it distinguishable from the V gesture (where the two sides must have approximatively the same length).

pigtail. The *pigtail* gesture is defined as follows:

$$pigtail = \begin{array}{l} LineRight, LineUpRight, LineUpLeft, \\ LineDownLeft, LineDownRight, LineRight \end{array}$$

In this case, the specification does not include any constraint in order to provide the user with a wide degree of freedom in the gesture proportions, according to the usual semantics of this kind of gesture.

left square. The *leftsquare* gesture is defined as follows:

$$leftsquare = \begin{array}{l} \left[LineLeft\right]_1, \left[LineDown\right]_2, \left[LineRight\right]_3 \\ with\ |distance\,(b_1, e_1) - distance\,(b_3, e_3)\,| \\ \leq 0.2 \cdot distance\,(b_1, e_1) \wedge \\ distance\,(b_1, e_1) \leq distance\,(b_2, e_2) \cdot 0.6 \end{array}$$

The formula constraints naturally reflect the fact that in a square bracket the two horizontal lines must be very similar in length ($|distance(b_1, e_1) - distance(b_3, e_3)| \leq 0.2 \cdot distance(b_1, e_1)$), and shorter than the vertical line ($distance(b_1, e_1) \leq distance(b_2, e_2) \cdot 0.6$).

left curly. The *leftcurly* gesture is defined as follows:

$$leftcurly = \begin{array}{l} \left[LineDownLeft\right]_1, \\ \left[LineDown, \left[LineDownLeft\right]_3, \right. \\ \left[LineDownRight\right]_4, LineDown\right]_2, \\ \left[LineDownRight\right]_5 \\ with\ |distance\,(b_1, e_1) - distance\,(b_5, e_5)\,| \\ \leq 0.2 \cdot distance\,(b_1, e_1) \wedge \\ |distance\,(b_3, e_3) - distance\,(b_4, e_4)\,| \\ \leq 0.2 \cdot distance\,(b_3, e_3) \wedge \\ distance\,(b_1, e_1) <= distance\,(b_2, e_2) \cdot 0.6 \end{array}$$

In the formula, the trajectory specification determines a left curly brace approximation as depicted in Figure 7(d). On the other hand, the constraints, according to the generic shape of a curly brace, require that the pairs of diagonal fragments (i.e., (A, F) and (C, D) in that figure) must be very similar in length, and that the upper and lower diagonal fragments (i.e., A and F) must be shorter than the central part of the gesture(i.e., informally, $B + C + D + E$).

zigzag. The *zigzag* gesture can be easily defined by composing two carets with a final diagonal line:

$$zigzag = caret, caret, LineUpRight$$

This specification does not require any dependency between the two carets' size and between the carets' size and the length of the final line, since in real zigzag trajectories such a strong regularity is not required and even difficult to obtain in the practice. On the other hand, each single caret maintains a certain degree of regularity thanks to the constraints included in its own specification.

arrowright. The *arrowright* gesture is defined as follows:

$$arrowright = \begin{array}{l} LineRight, \left[LineUpLeft\right]_1, \\ \left[LineDownRight\right]_2, \left[LineDownLeft\right]_3 \\ with\ |distance(b_1, e_1) - distance\,(b_2, e_2)\,| \\ \leq 0.2 \cdot distance\,(b_1, e_1) \wedge \\ |distance\,(b_1, e_1) - distance\,(b_3, e_3)\,| \\ \leq 0.2 \cdot distance\,(b_1, e_1) \end{array}$$

In the formula, the constraints reflect the fact that in an arrow the head must be barycentric, and this is obtained by requiring that the three lines composing it (i.e., A, B, C in Figure 7(e)) should have a very similar length.

X. The X gesture is defined as follows:

$$X = \begin{array}{l} \left[LineDownRight\right]_1, LineUp, \left[LineDownLeft\right]_2 \\ with\ |distance\,(b_1, e_1) - distance\,(b_2, e_2)\,| \\ \leq 0.2 \cdot distance\,(b_1, e_1) \end{array}$$

In order to match the common "X" shape, the specification only requires that the two diagonal lines must have a very similar length.

delete. To appropriately model this gesture, we can take advantage of another qualitative issue of our formalism, that is the ability to easily derive new gestures as 45-degrees-rotated variants of other gestures.

To clarify this aspect we can refer to Figure 8 showing how each basic gesture has a corresponding rotated variant, e.g., *LineDownRight* corresponds exactly to *LineRight* rotated by 45 degrees clockwise. Of course, this is accomplished at lower level by appropriately manipulating the corresponding underlying spatial relations. As a matter of fact, complex gestures, which are obtained composing the basic ones, can be in turn rotated by simply rotating their components.

Indeed, in this case, the *delete* gesture can be obtained from the X one through two steps, as shown in Figure 7(f). In the former step, the X is rotated 90 degrees clockwise (i.e., twice 45 degrees in Figure 8), obtaining the following gesture specification:

$$\begin{array}{l} \left[LineDownLeft\right]_1, LineRight, \left[LineUpLeft\right]_2 \\ with\ |distance(b_1, e_1) - distance(b_2, e_2)| \\ \leq 0.2 \cdot distance(b_1, e_2) \end{array}$$

Then, in the latter step, the above gesture is mirrored on the y axis, producing the final specification for the *delete* gesture:

$$delete = \begin{array}{l} \left[LineDownRight\right]_1, LineLeft, \left[LineUpRight\right]_2 \\ with\ |distance(b_1, e_1) - distance(b_2, e_2)| \\ \leq 0.2 \cdot distance(b_1, e_1) \end{array}$$

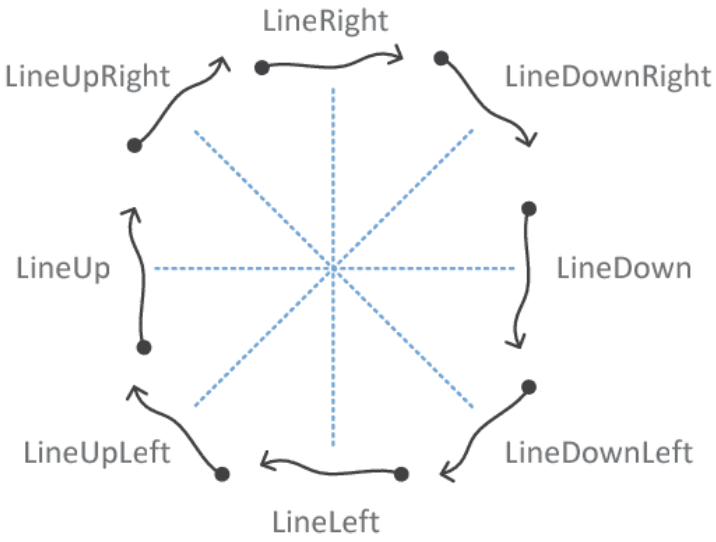


Figure 8 Basic gesture rotations.

	M(y)	M(x)	R(45)	R(90)	R(135)	R(180)	R(225)	R(270)	R(315)
Left	Right	Left	UpLeft	Up	UpRight	Right	DownRight	Down	DownLeft
Right	Left	Right	DownRight	Down	DownLeft	Left	UpLeft	Up	UpRight
Up	Up	Down	UpRight	Right	DownRight	Down	DownLeft	Left	UpLeft
Down	Down	Up	DownLeft	Left	UpLeft	Up	UpRight	Right	DownRight
UpLeft	UpRight	DownLeft	Up	UpRight	Right	DownRight	Down	DownLeft	Left
UpRight	UpLeft	DownRight	Right	DownRight	Down	DownLeft	Left	UpLeft	Up
DownLeft	DownRight	UpLeft	Left	UpLeft	Up	UpRight	Right	DownRight	Down
DownRight	DownLeft	UpRight	Down	DownLeft	Left	UpLeft	Up	UpRight	Right

Figure 9 Basic gesture transformations.

As a final consideration, Figure 9 shows the overall admissible transformations (i.e., symmetries and rotations) applied to our set of basic gestures. In the table, gesture names are shortened omitting the initial "Line" term (e.g., *Left* stands for *LineLeft*), moreover $M(x)$ and $M(y)$ represent mirroring on the x and y axis, respectively, whereas $R(\alpha)$ represents a clockwise rotation of α degrees. For example, the $LineUpLeft$ gesture, if mirrored on the x axis (i.e., the $M(x)$ column in the table) becomes $LineDownLeft$, whereas if rotated by 225 degrees clockwise (i.e., the $R(225)$ column in the table) becomes $LineDown$.

By applying such transformations, it is possible to easily derive the other three missing $1 gestures, i.e., *rightsquare, rightcurly* and *arrowleft,* since they are symmetric gestures of *leftsquare*, *leftcurly* and *arrowright*, respectively.

4.3 Towards Gesture Recognition

We are working on a gesture recognition technique based on our qualitative formalism in order to obtain a complete framework for gesture specification and recognition. To this aim, we have developed a Java support tool that we are currently experimenting on $1 gestures.

In particular, this tool includes a complete object model covering the gesture notation of our formalism, so that gesture specification formulas as the ones presented in the paper can be easily encoded and manipulated in Java, and a prototype gesture matching algorithm, which has been built on the underlying regular expression gesture encoding style.

The tool front-end allows the user to draw a gesture, internally sampled as a timed graphical object trajectory that can be later shown on the tool interface and/or recorded for further elaboration. For instance, Figure 10 and Figure 11 show the tool interface where the user has drawn an arrowright-like and a star-like gesture, respectively, in the upper left pane. These gestures have been encoded as the trajectories shown in the upper right pane, where the squares represent the sampled positions of the graphical object. As shown, the tool correctly recognized both the gestures, whose name is reported in the green box below the upper right pane together with the recognition confidence, in both cases 100%. Moreover, the rightmost pane shows a tree representation of the object model for the gestures currently loaded in the tool library, where the structure of the recognized gesture is highlighted in green.

Finally, the bottom part of the interface shows the complete results of the recognition phase, i.e., the similarity ratio between the drawn gesture and the other gestures in the library. In detail, for each gesture (named in the first column), the table reports the *path similarity ratio* (i.e., how much of the regular expression corresponding to the gesture matches the drawn path), the *constraint satisfaction ratio* (i.e., how many of the gesture trajectory constraints are satisfied), the derived *total match ratio* and the *time* requiredfor the overall matching process. As an example, in the case of the star gesture shown in Figure 11 the time required is 0.376 milliseconds.

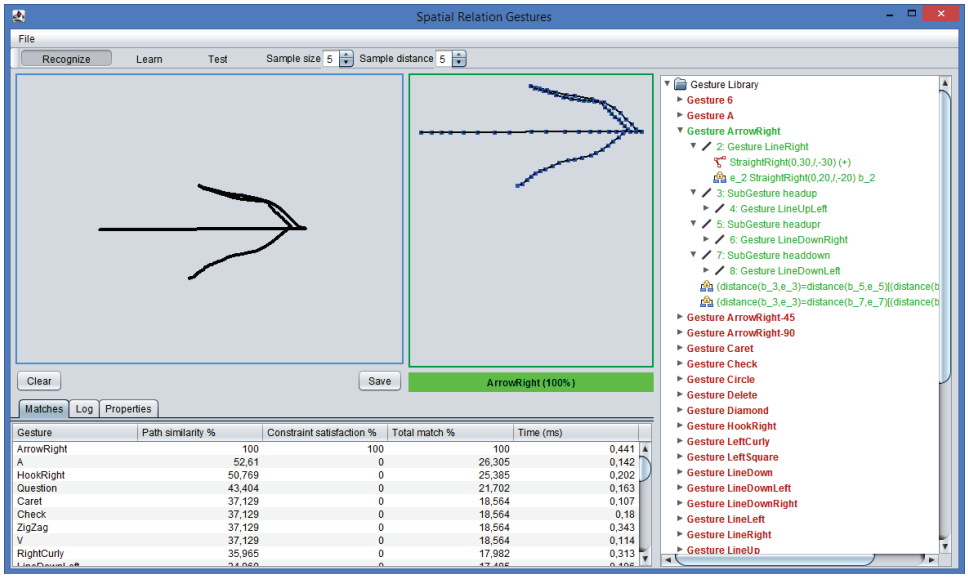


Figure 10 The interface: recognition of the *arrowright* gesture.

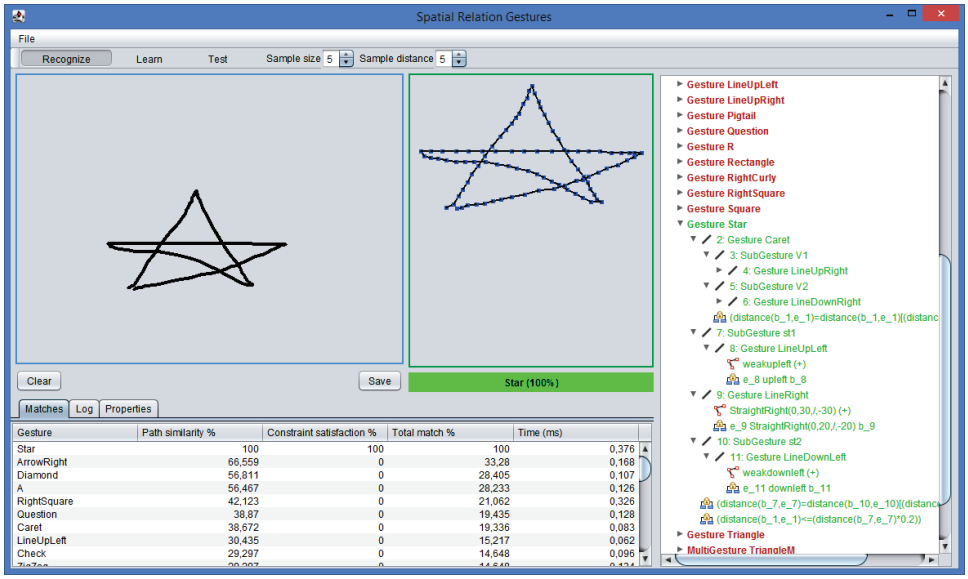


Figure 11 The interface: recognition of the *star* gesture.

5. CONCLUSIONS AND FURTHER RESEARCH

In this paper we have extended the spatial relation formalism presented in [1] with the notion of time in order to represent *spatio-temporal knowledge*. As a first application, we exploited the augmented formalism to support a qualitative specification of gestures, which is a novelty in the spatial relation literature.

The resulting technique provides gesture specification with a systematic and formal foundation. In particular, it supports a very natural gesture encoding style which is similar to the classic regular expression notation, making it easy to understand and manipulate. In fact, the qualitative nature of our framework makes it able to easily address a variety of relevant aspects like noise tolerance, scale- and rotation-invariance, compositionality and gesture derivation by rotation or symmetry.

Further research will be oriented both to theoretical and practical issues. First of all, we intend to refine the framework in order to apply spatial relations to general "evolving" graphical objects, i.e., objects which may change over time in different aspects like position, shape, colour, content, etc., and therefore not only moving objects as treated in the paper. In this way, the new time-extended formalism would open to a variety of challenging applications. Among these, we intend to investigate how to apply this framework in order to perform Visual Information Extraction on dynamic visual documents, e.g., web pages whose content changes over time or biomedical images acquired in successive phases (starting from preliminary works in [38, 39]).

On the other hand, in the gesture context, we intend to exploit our formalism to specify more complex gestures than the ones in the $1 set (see Section 5.1 for an early sketch on this aspect) and to refine the prototype tool shown in Section 4.3, in particular its recognition algorithm, in order to compare our overall qualitative gesture specification/recognition framework with others, starting from our inspiring approaches (i.e., [11],

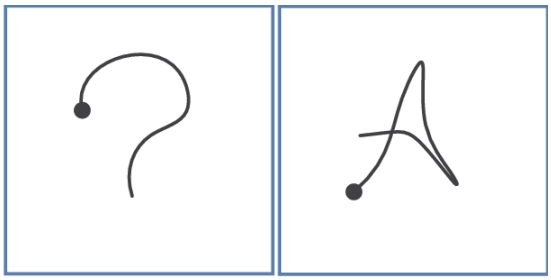


Figure 12 Examples of punctuation and alphabetic gestures.

[40], [41] and [42]), also with respect to aspects like speed and accuracy of recognition.

5.1 Beyond $1

We are investigating a number of complex gestures to test the expressive power of our specification method. As an example, so far we have already completed the specifications of some gestures within two classes partially addressed by $1, i.e., alphabetic letters and punctuation marks. For instance, let us briefly show the specification of the *A* and *question* gestures.

question. The *question* gesture (see Figure 12(a)) is defined as follows:

$$question = \begin{array}{l} \left[LineUpRight, LineDownRight\right]_1, \\ \left[LineDownLeft, Down\right]_2 \\ with\ distance(b_2, e_2) \geq distance(b_1, e_1) \cdot 1.2 \end{array}$$

In the formula, the constraint (i.e., $distance(b_2, e_2) \leq distance(b_1, e_1) \cdot 1.2$) requires the descending part of the question mark to have a length slightly greater than the width of its top part.

A. The gesture corresponding to this upper-case letter (see Figure 12(b)) is defined as follows:

$$A = \begin{array}{l} \left[LineUpRight, \right]_1, \left[LineDownRight\right]_2, \left[LineUpLeft\right]_3, \\ \left[LineLeft\right]_4 \\ with\ |distance(b_1, e_1) - distance(b_2, e_2)| \\ \leq 0.2 \cdot distance(b_1, e_1) \wedge \\ |distance(b_2, e_2) \cdot 0.5 - distance(b_3, e_3)| \\ \leq 0.2 \cdot distance(b_3, e_3) \wedge \\ distance(b_4, e_4) > distance(b_1, e_2) \cdot 0.5 \end{array}$$

This time, the constraints require that, respectively

- the two vertical sides of the letter are of nearly identical length,
- the horizontal line is more or less in the middle of the overall gesture,
- the horizontal line length is at least half of the letter base, that is the distance between the beginning of the left side and the end of the right one.

REFERENCES

1. Giuseppe Della Penna, Daniele Magazzeni, and Sergio Orefice. *A formal framework to represent spatial knowledge*. Knowledge and Information Systems, 51(1):311–338, 2017.
2. Giuseppe Della Penna, Daniele Magazzeni, and Sergio Orefice. *A spatial relation-based framework to perform visual information extraction.* Knowledge and Information Systems, 30(3):667–692, 2012.
3. Giuseppe Della Penna, Daniele Magazzeni, and Sergio Orefice. *Visual extraction of information from web pages.* Journal of Visual Languages and Computing, 21(1):23–32, 2010.
4. Giuseppe Della Penna, Daniele Magazzeni, and Sergio Orefice. *A general theory of spatial relations to support a graphical tool for visual information extraction.* Journal of Visual Languages and Computing, 24(2):71 – 87, 2013.
5. David Rempel, Matt J. Camilleri, and David L. Lee. The design of hand gestures for human–computer interaction: Lessons from sign language interpreters. *International Journal of Human-Computer Studies*, 72(10–11):728–735, 2014.
6. Ke Liu, Dunwei Gong, FanlinMeng, Huanhuan Chen, and Gai-Ge Wang. Gesture segmentation based on a two-phase estimation of distribution algorithm. *Information Sciences*, 394–395:88 – 105, 2017.
7. Vicente Nacher, Javier Jaen, Elena Navarro, Alejandro Catala, and Pascual Gonzalez. Multi-touch gestures for pre-kindergarten children. *International Journal of Human-Computer Studies*, 73:37 – 51, 2015.
8. Yu Huang, Guangyou Xu, and Yuanxin Zhu. Extraction of spatial-temporal features for vision-based gesture recognition. *Journal of Computer Science and Technology*, 15(1):64–72, Jan 2000.
9. Nem Khan Dim and Xiangshi Ren. Designing motion gesture interfaces in mobile phones for blind people. *Journal of Computer Science and Technology*, 29(5):812–824, Sep 2014.
10. Jin-Kai Zhang, Cui-Xia Ma, Yong-Jin Liu, Qiu-Fang Fu, and Xiao-Lan Fu. Collaborative interaction for videos on mobile devices based on sketch gestures. *Journal of Computer Science and Technology*, 28(5):810–817, Sep 2013.
11. Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 159–168, New York, NY, USA, 2007. ACM.
12. Max J. Egenhofer and Robert D. Franzosa. Point set topological relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991.
13. Max J. Egenhofer, David M. Mark, and John Herring. The 9-intersection: Formalism and its use for natural-language spatial predicates. Technical Report 94-1, National Center for Geographic Information and Analysis, 1994.

14. Eliseo Clementini and Paolino Di Felice. A comparison of methods for representing topological relationships. *Information Sciences - Applications*, 3(3):149 – 178, 1995.
15. Zhi-NongZhong, Ning Jing, Luo Chen, and Qiu-Yun Wu. Representing topological relationships among heterogeneous geometry-collection features. *Journal of Computer Science and Technology*, 19(3):280–289, 2004.
16. David A. Randell, Zhan Cui, and Anthony G. Cohn. A spatial logic based on regions and connection. In Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, October 25-29, 1992., pages 165–176, 1992.
17. Sanjiang Li and Mingsheng Ying. Region connection calculus: Its models and composition table. *Artificial Intelligence*, 145(1-2):121 – 146, 2003.
18. A. G. Cohn, D. A. Randell, and Z. Cui. Taxonomies of logically defined qualitative spatial relations. *International Journal of Human-Computer Studies*, 43(5-6):831–846, December 1995.
19. ReinhardMoratz. Representing relative direction as a binary relation of oriented points. In ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings, pages 407–411, 2006.
20. Till Mossakowski and ReinhardMoratz. Qualitative reasoning about relative direction of oriented points. *Artif. Intell.*, 180-181:34–45, 2012.
21. ReinhardMoratz and Marco Ragni. Qualitative spatial reasoning about relative point position. *J. Vis. Lang. Comput.*, 19(1):75–98, 2008.
22. Andrew U. Frank. Qualitative spatial reasoning with cardinal directions. In *Proc. 7th Austrian Conference on Artificial Intelligence, ÖGAI-91, Wien, 24.-27. September 1991*, pages 157–167, 1991.
23. Christian Freksa. Using orientation information for qualitative spatial reasoning. In *Spatio-Temporal Reasoning*, pages 162–178, 1992.
24. SpirosSkiadopoulos and ManolisKoubarakis. Composing cardinal direction relations. *Artificial Intelligence*, 152(2):143 – 171, 2004.
25. RoopK. Goyal and MaxJ. Egenhofer. Similarity of cardinal directions. In ChristianS. Jensen, Markus Schneider, Bernhard Seeger, and VassilisJ. Tsotras, editors, *Advances in Spatial and Temporal Databases*, volume 2121 of *Lecture Notes in Computer Science*, pages 36–55. Springer Berlin Heidelberg, 2001.
26. Isabel Navarrete, Antonio Morales, Guido Sciavicco, and M. Antonia Cardenas-Viedma. Spatial reasoning with rectangular cardinal relations. *Annals of Mathematics and Artificial Intelligence*, 67(1):31–70, 2013.
27. Philippe Balbiani, Jean-François Condotta, and Luis Fariñasdel Cerro. A model for reasoning about bidemsional temporal relations. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998.*, pages 124–130, 1998.
28. James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.
29. Markus Schneider, Tao Chen, Ganesh Viswanathan, and Wenjie Yuan. Cardinal directions between complex regions. *ACM Trans. Database Syst.*, 37(2):8:1–8:40, June 2012.
30. Sanjiang Li and Weiming Liu. Cardinal directions: a comparison of direction relation matrix and objects interaction matrix. *International Journal of Geographical Information Science*, 29(2):194–216, 2015.
31. Juan Chen, Anthony G. Cohn, Dayou Liu, Shengsheng Wang, Jihong Ouyang, and Qiangyuan Yu. A survey of qualitative spatial representations. *The Knowledge Engineering Review*, 30:106–136, 1 2015.
32. Timothy K. Shih. On computing temporal/spatial relations. *Information Sciences*, 107(1–4):37 – 61, 1998.
33. Brandon Bennett, Anthony G. Cohn, Frank Wolter, and Michael Zakharyaschev. Multi-dimensional modal logic as a framework for spatio-temporal reasoning. *Applied Intelligence*, 17(3):239–251, 2002.
34. VineethaBettaiah and Ramazan S. Aygun. Query-by-gaming: Interactive spatio-temporal querying and retrieval using gaming controller. *Journal of Visual Languages & Computing*, 29:63 – 76, 2015.
35. Shyamanta M. Hazarika. Qualitative Spatio-Temporal Representation and Reasoning: Trends and Future Directions. IGI Global, 2012.
36. Hans W. Guesgen, Gérard Ligozat, Jochen Renz, and Rita V. Rodríguez. Spatial and temporal reasoning. *Spatial Cognition & Computation*, 8(1-2):1–3, 2008.
37. Gerard Ligozat and Jean-Francois Condotta. On the relevance of conceptual spaces for spatial and temporal reasoning. *Spatial Cognition & Computation*, 5(1):1–27, 2005.
38. *Authors*, 2016.
39. Mohammad Ali ZareChahooki and NasrollahMoghaddam-Charkari. Bridging the semantic gap for automatic image annotation by learning the manifold space. *International Journal of Computer Systems Science and Engineering*, 30 (4), 2015.
40. Yang Li. Protractor: a fast and accurate gesture recognizer. In Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, Atlanta, Georgia, USA, April 10-15, 2010, pages 2169–2172, 2010.
41. Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. Gestures as point clouds: A $p recognizer for user interface prototypes. In *Proceedings of the 14th ACM International Conference on Multimodal Interaction*, ICMI '12, pages 273–280, New York, NY, USA, 2012. ACM.
42. Vittorio Fuccella and Gennaro Costagliola. Unistroke gesture recognition through polyline approximation and alignment. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 3351–3354, New York, NY, USA, 2015. ACM.